Polytech App3, Algorithmique

Mini Projet : Motus

Consignes

Ce TP est un mini-projet **optionnel**, et est à rendre au plus tard le 19/05/2024 à 23h59. Les rendus sont à envoyer à l'adresse `fpirot@lisn.fr`, et sont à faire en binôme. Ils se composent obligatoirement d'un fichier nom1_nom2.py pouvant se lancer depuis un terminal et exécuter une démo complète du projet, et d'un fichier nom1_nom2.pdf contenant un rapport expliquant le fonctionnement du code et justifiant les choix d'implémentation le cas échéant. Seuls les rendus complets (traitant les 3 exercices) seront considérés, et permettront d'obtenir un bonus maximum de 2 points à la note d'examen.

Le but de ce TP est d'écrire un programme permettant de jouer au célèbre jeu MOTUS. Ce jeu s'inspire du principe du Mastermind : le joueur doit deviner un mot dont on lui donne la longueur et la première lettre. Chacune de ses propositions doit être un mot valide (qui existe dans le dictionnaire, qui a la bonne longueur, et commence par la bonne lettre), et on lui indique les lettres bien placées et mal placées.

Exemple de déroulement d'une partie de MOTUS (le mot à deviner est SURPRISES).

Exercice 1 (Une interface dans le terminal)**.**
Commençons par implémenter une interface fonctionnant dans le terminal, et permettant de jouer au jeu MOTUS.

1. Télécharger le dictionnaire à l'adresse `www.lri.fr/~fpirot/mot/scrabble.txt`, puis stocker les mots de taille comprise entre 4 et 12 dans une liste dico.
2. Écrire une fonction randomWord(dico) de complexité $O(1)$ qui renvoie un élément aléatoire uniforme du dictionnaire dico donné en paramètre (cf. module random de Python).

La fonction randomWord a un biais concernant la longueur du mot qu'elle renvoie ; les mots de longueur 10 sont privilégiés car plus nombreux. On souhaite maintenant avoir le contrôle sur la longueur du mot choisi.

3. Nous allons cette fois-ci utiliser une structure de *dictionnaire* pour stocker les mots autorisés dans la variable dico. En python, un dictionnaire est une structure de données permettant d'indexer les valeurs selon une clé de n'importe quel type muni d'un ordre.

Typiquement, un tableau de taille n pourrait être simulé par un dictionnaire dont les clés seraient $0, 1, \ldots, n-1$. Nous allons associer à chaque clé ℓ la liste des mots de longueur ℓ, dont on note la taille n_ℓ.

Par exemple, on pourrait avoir le dictionnaire réduit suivant.

```
dico = {
    4 : ['kiwi'],
    5 : ['peche', 'poire', 'pomme', 'prune'],
    6 : ['banane', 'litchi'],
    9 : ['mirabelle']
}
```

4. Écrire une fonction randomWord(dico,ℓ) de complexité $O(1)$ qui renvoie un mot de taille ℓ aléatoire uniforme du dictionnaire dico donné en paramètre.
5. Écrire une fonction inside(w,lst) qui fait une recherche dichotomique pour vérifier si le mot w est dans la liste triée de mots lst, de complexité $O(\log |\text{lst}|)$.
6. Soit x le mot à deviner, de longueur ℓ. Écrire une fonction isValid(w,x,dico) qui détermine si w est une proposition valide (donc si w a la bonne longueur et la bonne première lettre, et s'il existe dans le dictionnaire), de complexité $O(\log n_\ell)$.
7. Soit x le mot à deviner de taille ℓ, et w le mot (valide) proposé par le joueur. Écrire une fonction compare(w,x) qui renvoie un tableau t de taille ℓ dont la valeur en position i est 2 si w[i] est bien placé, 1 si w[i] est mal placé (attention à ne pas utiliser plusieurs fois la même lettre de x !), et 0 sinon.

   Aide. On pourra utiliser une variable de type Counter (diponible dans le module collections de Python) pour compter les occurrences de chacune des lettres de x.
8. Utiliser les fonctions suivantes pour afficher le résultat de compare(w,x) de manière graphique : les lettres bien placées sont surlignées en rouge, et celles mal placées sont surlignées en vert (le terminal n'aime pas le jaune).

```
import colorama
colorama.init()

RED = colorama.Back.RED
GREEN = colorama.Back.GREEN
END = colorama.Back.RESET

def printRed(a) :
    print(RED+a+END,end='')

def printGreen(a) :
    print(GREEN+a+END,end='')
```

9. Écrire une fonction partie(n) qui lance une partie en n essais.
   - Le programme choisit un mot x à faire deviner, et affiche sa longueur et sa première lettre.
   - Pour chaque proposition du joueur, le programme vérifie que le mot est valide (autrement il demande une nouvelle proposition sans incrémenter le nombre d'essais effectués), puis affiche les lettres bien et mal placées.

— Le programme s'arrête quand le joueur a trouvé le mot, ou bien s'il a épuisé ses essais. Il termine en affichant le résultat de la partie.

Exercice 2 (Résolution automatique)**.**
Le but de cet exercice est de mettre au point un programme de résolution automatique pour le jeu MOTUS. On adoptera pour cela une stratégie naïve : à chaque étape, le programme fera une proposition aléatoire parmi l'ensemble des solutions possibles compte-tenu des informations qu'il aura obtenues lors de ses précédents essais.

1. Touver une solution pour accéder rapidement à la liste des propositions valides parmi un dictionnaire dico étant donné un mot x à deviner. Avec la bonne structure de données pour dico, cela peut se faire en temps $O(1)$.
2. Écrire une fonction update(words,w,comp) qui renvoie la liste des mots x parmi ceux de la liste words tels que compare(w,x) = comp.
3. Écrire une fonction partieNaive(x) qui joue automatique à MOTUS avec x comme mot à deviner, et renvoie le nombre de tentatives effectuées.

Exercice 3 (Une intelligence artificielle)**.**
Le but de cet exercice est de mettre au point une intelligence artificielle ayant de meilleures performances pour jouer à MOTUS, par rapport à l'approche naïve de l'exercice précédent. Les joueurs professionnels savent qu'ils peuvent avoir intérêt à faire une proposition qui n'est pas une solution possible si celle-ci leur permet d'obtenir davantage d'information. Par exemple, si l'on sait que le mot à deviner est de la forme CHA_EAU, alors les solutions possibles sont {CHAMEAU, CHAPEAU, CHATEAU}. Il est possible de déduire la solution en un seul essai, avec par exemple la proposition COMPTER, qui contient les lettres MPT et permettra de savoir laquelle des trois apparaît dans la solution.

Le principe de base d'une IA dont le but est la prise de décision est d'associer à chaque proposition possible un score, que l'IA cherche à maximiser (ou à minimiser). La définition de ce score est ce que l'on appelle une *heuristique*. Dans notre cas, nous allons chercher à maximiser la quantité d'information que l'on espère obtenir en faisant notre proposition.

1. On suppose qu'après i essais, l'ensemble des solutions possibles est S. Pour tout $w \in$ dico et $x \in S$, on note $S[w,x]$ l'ensemble des solutions possibles après avoir proposé le mot w au $(i+1)$-ème essai, si le mot à deviner est x. Le *gain* associé est $\log \frac{|S|}{|S[w,x]|}$, que l'on peut interpréter comme le nombre de bits de la solution que l'on peut déduire suite à la proposition w.

   En tant qu'heuristique, l'IA va proposer le mot w qui maximise un score $\sigma_S(w)$, défini selon une des règles suivantes.

   (a) $\sigma_S^0(w) \coloneqq \dfrac{1}{|S|} \displaystyle\sum_{x \in S} \log \frac{|S|}{|S[w,x]|}$ (gain moyen) ;

   (b) $\sigma_S^1(w) \coloneqq \min\limits_{x \in S} \log \dfrac{|S|}{|S[w,x]|}$ (gain minimum).

   Afin d'optimiser la complexité du calcul de $\sigma_S(w)$, on peut faire plusieurs remarques.

   (i) Pour tout $x' \in S[w,x]$, on a $S[w,x'] = S[w,x]$. Donc les ensembles $S[w,x]$ peuvent être calculés en partitionnant S selon le résultat de compare(w,x) pour $x \in S$. Notons $(S_1, \ldots, S_{r_w})$ cette partition.

   (ii) Maximiser $\sigma_S^0(w)$ est équivalent à minimiser $\tilde{\sigma}_S^0(w) \coloneqq \displaystyle\sum_{i=1}^{r_w} |S_i| \log |S_i|$.

(iii) Maximiser $\sigma_S^1(w)$ est équivalent à minimiser $\tilde{\sigma}_S^1(w) \coloneqq \max_{i \in [r_w]} |S_i|$.

Écrire une fonction choice(dico,heuristique=0) qui retourne la proposition de l'IA lorsque dico est une liste qui contient l'ensemble des solutions possibles, en suivant l'heuristique σ^0 ou σ^1.

Aide. Il est possible d'utiliser des tuples comme clés dans un dictionnaire.

2. Écrire une fonction partieIA(x,heuristique=0) qui joue à MOTUS avec une des heuristiques proposées lorsque le mot à deviner est x, et renvoie le nombre de tentatives effectuées.
3. Comparer les performances entre les deux heuristiques et la résolution naïve sur des ensembles de mots x choisis aléatoirement.

   Conseil. Stocker pour chacune des heuristiques le premier mot proposé en fonction de ℓ et de la première lettre, afin d'éviter d'avoir à refaire ce calcul très coûteux à chaque partie.